Model Based Reinforcement Learning with Graph Interaction Networks

Ashutosh Kakadiya
CS18S013
Department of Computer Science
IIT Madras
Chennai-36
cs18s013@smail.iitm.ac.in

Yogesh Tripathi
CS16B044
Department of Computer Science
IIT Madras
Chennai-36
cs16b044@smail.iitm.ac.in

Rohan Saphal
Mentor for the project
IIT Madras
Chennai-36

Abstract

Planning is a natural phenomena in human mind, where mind improves the policy for reaction in the real environment based on the model of the environment in the mind. Model-Based Reinforcement Learning is centered at making agents build a sufficiently good model of the environment and then plan on the model. Dyna is an architecture which integrates learning, planning and reacting in the agent. In this work, we use the Dyna architecture in Model-based Reinforcement Learning after adding a prior knowledge about the structure of the environment in the agent's model. This is realized by capturing interactions within different parts of the agent and environment using graphs. The injection of this prior information into the model makes model-learning faster, with better sample efficiency and hence planning stabilizes early in the Dyna architecture.

1 Introduction

Model based Reinforcement Learning allows the agent to plan and react. The model can augment the learning in several ways. The agent learns to solve the control problem in much less environment samples[4]. In this work, we examine if we can learn better model in lesser environment sample if we inject some prior knowledge about the environment-agent structure and their interactions. We inject this knowledge by capturing these interactions in the form of a graph. The aim of the study is to learn a good model to plan on, in much lesser real samples by using the graph-based model.

Interaction Network creates a model of the environment by capturing interactions between different parts of the agent and environment as graph. Dyna integrates planning, learning and reaction by interleaving the steps of model learning and solving the control problem. Trust Region Policy Optimization(TRPO) by Schulman et al.[**?**]- A policy optimization technique has shown good results in most of the environments and is by far the most trusted optimization technique. Here our goal is to combine all these frameworks to obtain an agent which uses interaction network to capture interactions between the agent and environment so as to learn model with lesser samples, which plans like in Dyna architecture and uses TRPO to optimize it's policy in each reaction and planning phase of Dyna.

2 Literature Review

The first paper that we studied on Model-based Reinforcement Learning for this work was on Dyna by Sutton [9]. In this paper, an architecture- Dyna is presented which integrates planning with an action model and determining an optimal reactive policy through reinforcement learning methods. The paper presents only the architecture and provides a lot of flexibility in terms of the implementation of various aspects of the architecture- like the reinforcement learning algorithm, or way to learn the model. That being said, the paper also avoids the issues in implementation of the same. How is the model learnt for large state spaces? How will instability training tackled in training the model using approximators? How will model be trained using supervised learning if supervised learning assumes i.i.d training samples whereas those generated from a trajectory are not i.i.d samples? How will a good exploration of the state, action space be ensured? All these questions are not clearly addressed in the paper.

In case of continuous action, learning the action-model is even more challenging which is not explored in the paper. Exploration becomes a bigger issue and it is impossible to learn an optimal policy by simply learning Q-function. The issue of learning a wrong model is also not discussed in the pa- per. The choice of ratio between number of samples from real environment to the number of planning steps is also not addressed, which is crucial for learning good policies. Ideas developed in the paper are on a very high level and concrete details about each aspect of the architecture is missed out. There are no experiments shown in the paper to provide evidence for the advantage of using Dyna. However, it does not change the fact that the paper does provide a novel framework for an agent to plan and react in an environment.

The next milestone paper in our work was based on Interaction Networks by Battaglia et al.[1]. An interaction network is simply a series of functions applied to a graph, represented as $G = (O, R)$ where O is object(node) representation and R is the edge representation. The output of $IN(G)$ corresponds to the next state of objects, O'. The function IN is defined as:

$$IN(G) = \phi_O(a(G, X, \phi_R(m(G))))$$

where ϕ_O and ϕ_R are MLP and a and m are aggregating and marshalling functions which are simple matrix operations to get representations of interactions. One thing we can notice is that there is a homogeneity assumed between all the interacting objects. With this framework, we cannot learn the interaction among dissimilar objects which have totally different state representations. This limits the usage of framework for many real world cases because usually we have different state representations available of different kinds of interacting objects based on measurable properties for the object.

The paper also uses one interaction term for each relation, which means we cannot represent n-th order relations or interactions that can have variable order as that would require f_R to be able to handle input of variable lengths. There is not much background given upon selecting the relations between objects. Identifying appropriate relations among the objects and characterizing them with right set of parameters is also a non-trivial task which is not well-stated in the paper. Selecting a complete graph for every problem is not a good option because some objects might not be interacting at all. Finer details like ensuring coverage of sufficiently large portion of state-space is not described in detail.

Then, we looked into some related papers on Graph-based Model in reinforcement learning. This includes a paper on Graph Networks as Learnable Physics Engines for Inference and Control by Alvaro Sanchez-Gonzalez et al.[6]. In this paper, Graph Network based forward models are presented along with inference models, both of which treat physical system as graphs, followed by control algorithm which use these forward and inference models for planning and policy learning. In this paper, the way of deciding the graph structure, selecting the objects and appropriate characterizations of relations is not discussed in detail. The paper also uses MPC planning to update the policy function and GN-based model simultaneously, which can cause divergence in the model. If GN-based model is inaccurate, then incorrect gradients will be propagated in while updating policy network. Thus, this approach does not take into account the common problem of model-based planners that the errors magnify more and more over long trajectory predictions. The model is also not successful in *Cheetah* environment because there is not a much of a common structure among various bodies and joints which breaks down the generalization.

Other related paper on graph based model that we looked at was on NerveNet by Wang et al.[11]. In this paper, a graph network based framework is presented, which is the NerveNet. It has an input model, a propagation model and an output model on top of which it learns the policy. The graph structure selection is clearly mentioned in this paper. The authors have clearly stated that they have used the same XML-based kinematic tree as organized in simulators such as MuJoCo engine. Between every pair of nodes, there is only one edge type allowed in the paper, hence structures like multigraphs are not explored. In the input model, same F_{in} is applied to each of the nodes, but this is a limitation of the input model as it treats each node similarly which might not be the case. In the propagation step, the message computed is sent across only to the neighbors in the graph. However, it might be helpful to send that messages to some nodes which indirectly depend on the sender node. In the message aggregation step, the aggregation function is suggested to be chosen a summation, average or max-pooling. However, all of these function will cause loss of information from the incoming messages and will also make the sender insensitive to which receiver provided which message. In case of learning the policy, the graph structure of the environment is not exploited. Using the graph structure might give better results and it would inject more prior information about interactions than simply using state vectors.

Then, we looked at various learning methods available to solve the control problem. This includes Deep Deterministic Policy Gradient methods(DDPG) by Lillicrap et al.[5], Proximal Policy Optimization(PPO) by Schulman et al.[8] and Trust Region Policy Optimization(TRPO) by Schulman et al.[7].

In DDPG, an actor critic approach is used to mitigate the issue of continuous action space learning. DDPG learns deterministic target policy using stochastic policy. DQN shows impressive performance in very high dimensional discrete action space like Atari games. The advantage in discrete action space is that we can easily learn optimal policy by following greedy approach on policy iteration but in continuous space it becomes very complex optimization problem and computationally expensive as in every iteration step, we have to perform optimization on very large action space, which is unfeasible in real environment dynamics. In DDPG there are two main building blocks, actor and critic which are both neural networks. Actor predicts the action from continuous action space based on current parameter θ and policy $\pi(a|s)$ and critic evaluates the current action determined by actor and provides feedback about it via TD Error. Actor takes input as current state and gives output as discrete action in the continuous action space. Critic gives output as simple Q values for current state and action determined by actor. After that gradients of actor and critic are updated based on TD error.

PPO introduces a new objective function called "Clipped surrogate objective function" that constrains the policy change in a small range using a clipping methods. In layman terms, the range of derived gradient will be restricted by some parameters to avoid high update, which introduces variability in training. Instead of using log probability in policy gradient, it takes ratio like importance sampling in off-policy between the probability of action under current policy to the probability of the action under previous policy. If this ratio is greater than 1, then the current action is more likely to be picked in current policy than in the old policy. But one problem arrives here, what if the ratio is very high. This will lead to big gradient steps in training which will make training unstable. To tackle this issue gradient clipping is used which clips the gradient by a hard threshold. Because of clipping, we can do training in parallel. PPO allows to run multiple epochs of gradient ascent on the samples without causing destructively large policy updates. This allows to squeeze more out of the data and hence reduce sample inefficiency.

TRPO improves the performance of DDPG as it introduces a surrogate objective function and a KL divergence constraint, guaranteeing non-decreasing long-term reward. TRPO is also a policy gradient method. It adopts the actor-critic architecture, but modifies how the policy parameters of the actor are updated. TRPO is a scalable algorithm for optimizing policies in reinforcement learning by gradient descent. Model-free algorithms such as policy gradient methods do not require access to a model of the environment and often enjoy better practical stability. Consequently, while straightforward to apply to new problems, they have trouble scaling to large, nonlinear policies. TRPO couples insights from reinforcement learning and optimization theory to develop an algorithm which, under certain assumptions, provides guarantees for monotonic improvement. It is now commonly used as a strong baseline when developing new algorithms. The objective function is also called a "surrogate" objective function as it contains a probability ratio between current policy and the next policy. TRPO successfully addresses the problem imposed by DDPG that the performance does not improve monotonically. The subset of region that lies within the constraint is called trust region. As

long as the policy change is reasonably small, the approximation is not much different from the true objective function. By choosing the new policy parameters which maximizes the expectation subject to the KL divergence constraint, a lower bound of the expected long-term reward is guaranteed, which disallows step size parameter to highly impact the TRPO's performance.

Other than these, a few more papers, whose work was studied include Neural Relational Inference for Interacting Systems by Kipf et al.[3]. n this paper, a variational auto-encoder is used to determine the graph structure of interaction between various objects from observed dynamics. The framework uses the encoder-decoder paradigm to attain its goal. The paper assumes that the dynamics can be modeled by a GNN given an unknown graph, $\vec{\boldsymbol{z}}$ where $\vec{\boldsymbol{z}}_{ij}$ represents discrete edge type between objects v_i and v_j. In the encoder portion, the specification of graph neural network is left very superficial. The choice for fully-connected MLP or 1D convolutional neural network is not justified. However, the two pass method, to ensure that information from whole graph, and not just neighbors, is used to predict the next state, is clearly justified. The curious form of $\vec{\boldsymbol{z}}_{ij}$ used while sampling is also not clearly stated as to why i.i.d samples from Gumbel(0,1) distribution is added to current state. The decoder portion assumes Markovian dynamics while generating the next set of states from the current set of states and graph structure, as it uses interaction network(IN). However, this may not hold for many applications. Also note that the conditional probability of next state given current state and graph is a normal distribution of a learnt $\vec{\boldsymbol{\mu}}$ from the IN and fixed diagonal covariance. Therefore, here all node features are assumed to be statistically independent which need not hold. Also, the variance estimate can also be made better by learning it from state vectors and aggregated messages, like $\vec{\boldsymbol{\mu}}$, instead of keeping it to be fixed throughout.

Other papers which were glossed over as a part of literature review include Graph Attention Networks by Veličković et al. [10] and VAIN: Attentional Multi-agent Predictive Modeling by Yedid Hoshen[2] both of which are based on bringing attention mechanism to identify important relations between different parts of agent and environment.

3 Model Based Reinforcement Learning

In model free reinforcement learning, we have used the most popular and successful method, so first we iteratively collected the data from given environment, then estimating the gradient of the policy and based on that improving the current policy and repeating this procedure to get the best policy for particular environment. We are not keeping the data. So, at every iteration the policy will be trained on completely new data, sampled from the environment. Wherever in model based reinforcement learning[9], the main aim is to make more extensive use of the data and reduce the sample complexity to learn the policy; we use that data to train the model which learn the real dynamics of real environment. This model will provide the samples on which policy can be learned and also gradient information. We assume here the model as a neural network function approximator.Policy also be represented by neural network to mitigate the scalability issues for very complex real world environment.

3.1 Model Learning

A standard Multi-layered feed forward neural network(MLFNN) is used to learn the transition dynamics of real environment. Training will be done by standard methods in deep learning like back-propagation. Model learning is supervised learning problem where input for the MLFNN is the current state and action, it will predict the next state. We denote the the model as a function approximator $\hat{f}_\phi$

The objective of the model is to find optimal parameters by minimizing the mean squared error loss:

$$\min_{\phi} \frac{1}{\mathcal{D}} \sum_{s_t, a_t, s_{t+1} \in \mathcal{D}} \left\| s_{t+1} - \hat{f}_\phi(s_t, a_t) \right\|^2 \tag{1}$$

Here, $\mathcal{D}$ is the dataset buffer which contained the samples which are already have experienced for learning policy. Adam optimizer is used as optimizer for described problem. The standard techniques are used to mitigate issues of over-fitting and gradient vanishing problem like batch normalization and normalizing input and output data for training the neural network.

3.2 Policy Learning

The main objective of the reinforcement learning is to maximize the expected sum of rewards given MDP, $\mathcal{M}$. As explained in algorithm 1 At the training time, first the model based will learn to approximate the given MDP $\mathcal{M}$ based on collected samples from given environment.So, model $\hat{f}_\phi$ learn from the samples. After learning the model, our agent (policy) will be trained on the fabricated samples, generated by model $\hat{f}_\phi$. And keep repeating this process until the policy shows good performance on real environment.

Algorithm 1 Vanilla Model-Based Deep Reinforcement Learning

1: Initialize policy π_θ and a model $\hat{f}_\phi$
2: Initialize an empty dataset $\mathcal{D}$
3: **loop**:
4: Collect samples from the real environment f using policy π_θ and add them into dataset $\mathcal{D}$
5: Train the model $\hat{f}_\phi$ using $\mathcal{D}$
6: **loop**:
7: Collect fabricated samples from $\hat{f}_\phi$ using π_θ
8: Update the policy on collected fabricated samples.

4 Graph Network Model Based Reinforcement Learning

Using the vanilla model based reinforcement learning, the sample complexity for learning the policy can be reduce in a great extent. But while training the model to learn the environment dynamics, it wont depict the any hidden structure, relations between states or how reward is based on states.

We present our novel Model based RL algorithm which incorporates the learning of interactions between the states in form of graph while learning the model dynamics to approximate $\hat{f}_\phi$. Basically these model will learn on real world data only but in the form of graphs called interaction network[1]. The rest procedure will be similar as Vanilla Model based RL. Formation of graph and interaction network are described in next subsection.

4.1 Interaction Network

Interaction Network is an architecture for reasoning about complex physical interactions between objects, called an Interaction Network. The model in its general form takes in object states and interactions between objects as input and simulates the system forward to produce next object states or to reason about physical quantities. The paper[1] proposes an implementation of the interaction network using deep neural networks, and discusses results on three simulated scenarios. Results described in paper[1] are promising and show good generalization to varying object shapes, sizes and physical properties.Interaction Network is designed to work on graphs where objects are nodes and edges are relations between objects. This is to make it scalable to different environments.

Interaction network has three main models : structured the models, simulation and Deep Learning. Structured the model will be different based on different objects as the dynamics will be varying. Interaction networks takes input as relations between different objects in a graph form, where each object represented by nodes and interaction between them by edges. So, it allows to learn important and potential interactions for different input data rather then considering learning all by imposing fixed architecture.

The basic IN is defined as,

$$IN(G) = \phi_o(a(G, X\phi_R(m(G)))) \tag{2}$$

$$\begin{aligned} m(G) = B = b_{kk} = 1..N_R \qquad & a(G, X, E) = C = c_{j_j} = 1..N_O \\ f_R(b_k) = e_k \qquad & f_O(c_j) = p_j \\ \phi_R(b_k) = e_k \qquad & f_O(c_j) = p_j \\ \phi_R(B) = E = \{e_k\}_{k=1..N_R} \qquad & \phi_O(C) = P = \{p_j\}_{j=1..N_O} \end{aligned} \tag{3}$$

Briefly discussing, the marshalling function, m, rearranges the objects and relations into interactions terms, b_k is tuple shows interaction between sender, receiver and their relation attributes. The relational model ϕ_R predicts the effect of each interaction. a is aggregation function which aggregates all effects, $e_k \in E$. ϕ_O is the object model that predicts, how the object and dynamics influence the objects for next timetemp t_{t+1}. In the following subsection we have described how to build the graph network for IN based on given environment.

4.2 Building Graph Model for IN

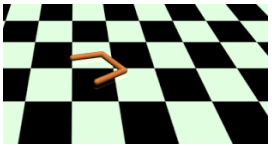

Figure 1: Swimmer MuJoCo Environment

Identifying the objects as nodes and the relations between them is one of the important and crucial thing in Interaction network. There is no global method for constructing the graph matrix. For, simplicity as mentioned earlier we assumed that graph is fully connected. The objects are mainly divided into two groups. Local objects and global objects. Local objects can influence only his neighbours, wherever Global objects will influence all the other objects. Global object's parameters are concatenated with all local object's parameters.

For example here we have done experiments on swimmer MuJoCo environment. In swimmer environment there are total 8 parameters, $\gamma, \dot{\gamma}, \theta_1, \dot{\theta_1}, \theta_2, \dot{\theta_2}, \dot{x}, \dot{y}$. Here $\dot{x}, \dot{y}$ are indicates the displacement in X and Y direction done by swimmer agent. γ is the yo and θ_1 and θ_2 are the angles between joints. So, any change in rate of angle will directly reflected on $\dot{x}$ and $\dot{y}$. Therefore considering them as global parameters we concatenated with local objects. So, here the graph matrix O will be of dimensions 4×3 as,

$$\begin{bmatrix} \gamma & \theta_1 & \theta_2 \\ \dot{\gamma} & \dot{\theta_1} & \dot{\theta_2} \\ \dot{x} & \dot{x} & \dot{x} \\ \dot{y} & \dot{y} & \dot{y} \end{bmatrix} \tag{4}$$

For every different environments, local and global objects will be decided manually. We can call it Human-in-Loop AI.

4.3 Interaction Network with Model Based RL

In this section, we are going to present our novel and simple method which combines the idea of Interaction network with model based Reinforcement learning. In vanilla model based RL, the model is directly learning to predict the next state and reward based on environment samples. Assumption on the samples are there all are I.I.D. Our intuition is if somehow we capture and incorporate silent relations between dynamics of given environment then the model can learn more rich presentations of environment dynamics.

Based on this intuitive approach, first we capture the interactions and relation between environment dynamics with the help of interaction network as described in section 4.1. And after using abstract information model is learned.The pseudo code is shown in Algorithm 2. First, we make a graph model based on certain assumptions on how different objects are connected with each other. We assumed fully connected graph network. So, every object is connect with every other objects. Here multi-layered neural network will be replaced by the interaction networks.Second, we use Trust Region Policy Optimization (TRPO) to optimize the policy over the model ensemble. Third, we use the model ensemble to monitor the policy's performance on validation data, and stops the current iteration when the policy stops improving.

Algorithm 2 Graph Interaction Network Model-Based Deep Reinforcement Learning

Initialize the graph interaction network ϕ for the given environment, to be used in model $\hat{f}_\phi$
Initialize policy π_θ
Initialize sample buffer, $\mathcal{D}$
loop:
    Collect samples from the real environment f using policy π_θ and add them to $\mathcal{D}$
    Update the policy on samples in $\mathcal{D}$.
    Train the model $\hat{f}_\phi$ [graph interaction network] using $\mathcal{D}$
    loop:
        Collect fabricated samples from $\hat{f}_\phi$ using π_θ
        Update the policy on collected fabricated samples.

5 Experiments

We divided the set of experiments to be performed in the following way. All experiments were performed on Swimmer environment of MuJoCo.

1. We first learned a model of the environment by using a multilayered perceptron(MLP) which takes state and action stacked, and returns the next state and reward prediction. We chose 2 hidden layers in this neural network with 1024-1024 nodes in the hidden layers. The number of epochs was fixed to be 200. The model was trained by back propagation with mini batch size of 100 and Adam update rule. Then, we compared the one-step and recursive loss obtained in the model.

2. We then learned a model of the environment by modelling the interaction between different parts of the agent and the model using graph. Then, we built a model using Interaction network framework and trained the two multilayered perceptrons, ϕ_O and ϕ_R using back propagation with mini batch size 100 and Adam update rule. The number of layers in both the MLP was fixed to be 1024-1024. D_E was chosen to be 3, which is the dimensionality of the abstract representation of each relation. Then, we compared the one-step and recursive loss obtained in this model against the model which used MLP without graph prior.

3. Then, we executed three different reinforcement learning algorithms- Deep Deterministic Policy Gradient(DDPG), Trust Region Policy Optimization(TRPO) and Proximal Policy Optimization(PPO) on model-free learning. We found that TRPO performs the best for the chosen environment and hence selected this for next set of experiments.

4. We finally conducted our last set of two experiments in which, in first case we ran a Dyna framework to learn the policy using TRPO algorithm on MLP model without graph structure and then, we ran the same experiments for the model with graph structure. We alternated between 50000 steps of model and environment samples for updating the policy and used a buffer of size 50000 for storing the environment samples for updating the model.

The graphs for all the experiments conducted are given below.

One step Graph based Model Prediction :

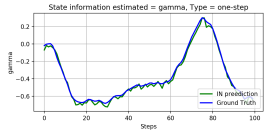

Figure 2: Feature γ one-step error.

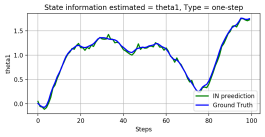

Figure 3: Feature θ_1 one-step error.

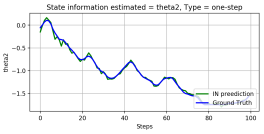

Figure 4: Feature θ_2 one-step error.

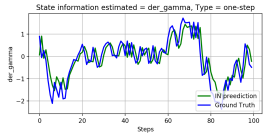

Figure 5: Feature $\dot{\gamma}$ one-step error.

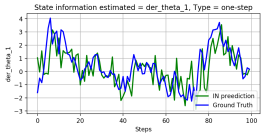

Figure 6: Feature $\dot{\theta_1}$ one-step error.

Figure 7: Feature $\dot{\theta_2}$ one-step error.

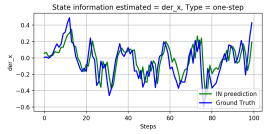

Figure 8: Feature $\dot{x}$ one-step error.

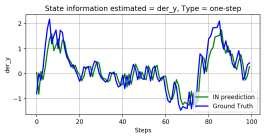

Figure 9: Feature $\dot{y}$ one-step error.

One step MLP based model prediction:

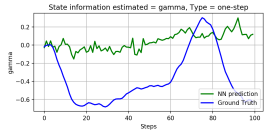

Figure 10: Feature γ one-step error.

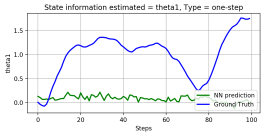

Figure 11: Feature θ_1 one-step error.

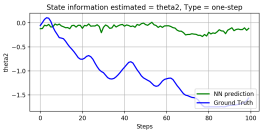

Figure 12: Feature θ_2 one-step error.

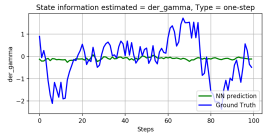

Figure 13: Feature $\dot{\gamma}$ one-step error.

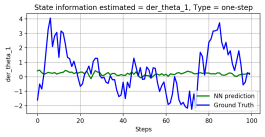

Figure 14: Feature $\dot{\theta_1}$ one-step error.

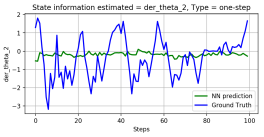

Figure 15: Feature $\dot{\theta_2}$ one-step error.

Figure 16: Feature $\dot{x}$ one-step error.

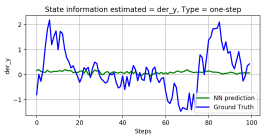

Figure 17: Feature $\dot{y}$ one-step error.

It is clearly seen that the graph based model learns a much better model than the MLP based model. It is much closer to the ground truth which is due to addition of prior knowledge about the interactions.

The comparison of reward obtained from model-free execution of DDPG, TRPO and PPO is as follows:

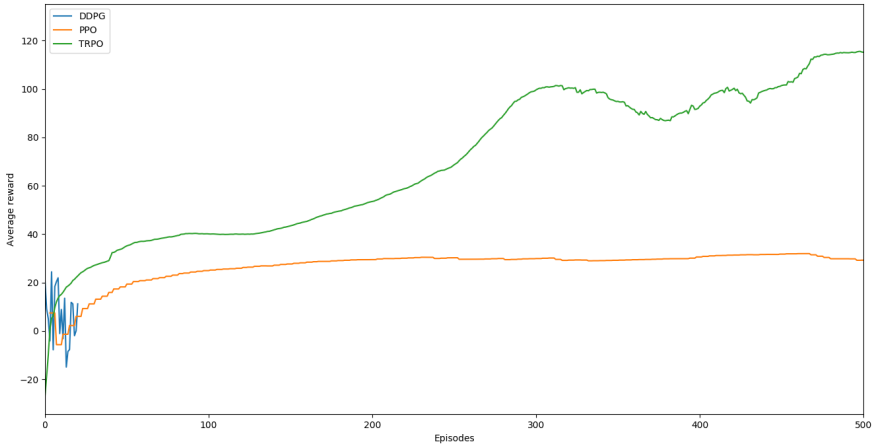

Figure 18: Comparison of average reward in TRPO, PPO and DDPG.

We see that TRPO performs better than DDPG and PPO on this environment, precisely why we selected TRPO for next experiment.

A comparison of Graph based model and MLP based model used with TRPO in Dyna architecture is shown below. (Note: Graph based model requires completion)

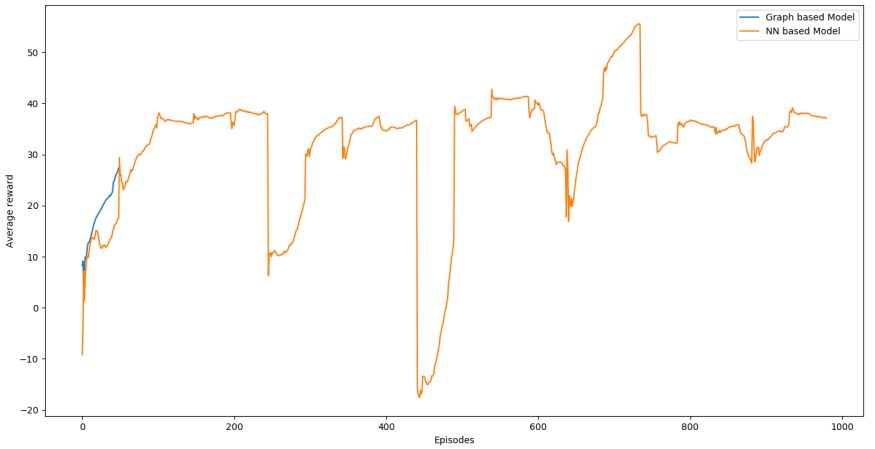

Figure 19: Comparison of average reward for Graph-based and MLP based model using TRPO in Dyna architecture.

6 Discussion and Future Work

The graph based model has shown very good performance in learning the dynamics of the environment than vanilla neural network. The main objective is to make the graph model for given environment. The identification of global parameters is very difficult. The assumption that all objects are interacting with each other is also not true in most cases and hence identifying important interactions between various parts of agent and environment and deciding the graph structure to use, still remains as a crucial problem to tackle.

We can also incorporate graph structure in the network for policy for improving learning. This should also take lesser learning time than vanilla neural network policy for continuous complex real world environments as giving the prior information about the environment structure allows each component in the graph to exploit this structure while choosing it's action component.

References

[1] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016.

[2] Yedid Hoshen. VAIN: attentional multi-agent predictive modeling. *CoRR*, abs/1706.06122, 2017.

[3] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2688–2697, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[4] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *CoRR*, abs/1802.10592, 2018.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[6] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *CoRR*, abs/1806.01242, 2018.

[7] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[9] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

[10] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[11] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks.